

JTRS HF ALE MAC API Service Definition

V1.0
December 15, 2000

Prepared for the
Joint Tactical Radio System (JTRS) Joint Program Office

Prepared by the
Modular Software-programmable Radio Consortium
Under Contract No. DAAB15-00-3-0001

Revision Summary

1.0	Initial release
-----	-----------------

Table of Contents

1	INTRODUCTION.....	1
1.1	OVERVIEW.....	1
1.2	SERVICE LAYER DESCRIPTION.....	1
1.3	MODES OF SERVICE.....	2
1.4	SERVICE STATES.....	2
1.5	REFERENCED DOCUMENTS.....	2
2	UUID.....	3
3	SERVICES.....	4
3.1	ESTABLISH CONNECTION.....	4
3.2	TRANSMIT AND RECEIVE SAMPLES.....	7
4	SERVICE PRIMITIVES.....	8
4.1	CONNECTIONCOMMANDS.....	8
4.1.1	connectionReq.....	8
4.1.2	initiateTransmit.....	9
4.1.3	terminateTransmit.....	9
4.1.4	disconnectReq.....	10
4.2	CONNECTIONSIGNALS.....	11
4.2.1	connection Confirm.....	11
4.2.2	connectionInd.....	12
4.2.3	disconnectInd.....	13
4.3	CONNECTION ESTABLISHED : TRANSMIT AND RECEIVE SAMPLES.....	13
4.4	MACPUSHPACKET.....	14
4.4.1	spaceAvailable.....	14
4.4.2	enableFlowControlSignals.....	15
4.4.3	enableEmptySignal.....	16
4.4.4	setNumOfPriorityQueues.....	16
4.4.5	getMaxPayLoadSize.....	17
4.4.6	getMinPayLoadSize.....	17
4.4.7	pushPacket.....	18
4.5	SIGNALS BB.....	19
4.5.1	signalHighWatermark.....	19
4.5.2	signalLowWatermark.....	19
4.5.3	signal Empty.....	20
4.6	COMMON STRUCTURES.....	20
4.6.1	Stream Control Structure.....	20
5	ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.....	22
6	PRECEDENCE OF SERVICE PRIMITIVES.....	30
7	SERVICE USER GUIDELINES.....	30

8	SERVICE PROVIDER-SPECIFIC INFORMATION.....	30
9	IDL.....	30
9.1	IDL FOR ALE CONFIGURE.....	30
9.2	IDL FOR ALE OPERATIONS.	33
9.3	IDL FOR ALE RESPONSES.....	35
9.4	IDL FOR CONNECTION.....	38
9.5	IDL FOR HF PACKET.....	39
9.6	IDL FOR HF API.	40
10	UML.	42

List of Figures

Figure 1.	API Scope	1
Figure 2.	Sequence Diagram for Service User Local Initiated Connection.....	6
Figure 3.	Incoming Connection Sequence Diagram.....	7
Figure 4.	connectionCommands.....	8
Figure 5.	Connection Signals	11
Figure 6.	Packet BB	14
Figure 7.	Signal BB.....	19
Figure 8.	Stream Control.....	21
Figure 9.	Stream Control Sequence Diagram.....	21

List of Tables

Table 1.	Cross-Reference of Services and Primitives.....	4
Table 2.	High Water and Low Water and Empty On.....	22
Table 3.	High Water and Low Water Off and Empty On.....	25
Table 4.	High Water and Low Water and Empty Off.....	27

1 INTRODUCTION.

1.1 OVERVIEW.

Automatic Link Establishment (ALE) is a means of automatically establishing a radio link between two or more HF stations. Radios using ALE still operate in the HF band and all characteristics of HF signal propagation still apply.

There are two major differences between conventional HF and ALE HF. Unlike conventional HF communication, ALE allows selective calling to other similarly equipped HF stations. ALE also automatically chooses the best available frequency from a preprogrammed list of frequencies to make the call. All that needs to be known is the address (ALE call sign) of other ALE stations with which communication is desired.

Establishing ALE communication is similar to placing calls using a telephone. An operator chooses a station address and starts the call. The ALE system automatically sets up a two-way communication link. Once a link is established, the HF ALE system operates the same as a conventional HF system.

In Figure 1 below, the "WF Specific Resource" may be one of many waveforms (e.g. 188-141, STANAG 5066, analog voice, etc.) but the same interface is provided at the MAC layer for all waveforms.

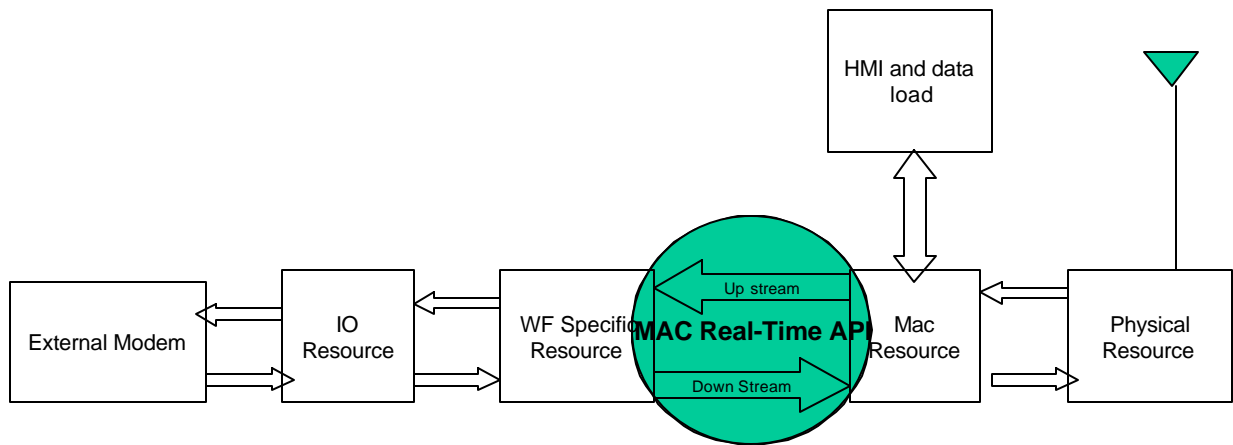


Figure 1. API Scope

1.2 SERVICE LAYER DESCRIPTION.

The HF ALE MAC Real-time API Service user is the waveform-specific resource which provides modulated digital samples to the HF ALE MAC Real-time API Service Provider. The Frequency tables are loaded through HF ALE MAC Non-Real-time Service Definition.

1.3 MODES OF SERVICE.

There are no specific Modes of Service.

1.4 SERVICE STATES.

Current State	Logical Event	Condition	Action	Next State
No Connection	Service User invokes <i>connectionReq</i>		Service Provider attempts to establish an ALE connection	Waiting for connection Confirm
	Service Provider invokes <i>connectionInd</i>		Service User provides the logical connections to waveforms	Connection Established/Receiving
Waiting for connection Confirm	Service Provider invokes <i>connectionInd</i>	ALE connection established	Service User provides the logical connections to waveforms	Connection Established/Receiving
		ALE connection failed		No Connection
Connection Established/Receiving	Service User invokes <i>disconnectReq</i>		Service User notifies waveforms of the lost connection.	Waiting for disconnection confirm
	Service Provider invokes <i>disConnectInd</i>		Service User notifies waveforms of the lost connection	No Connection
	Service User invokes <i>initiateTransmit</i>		Service Provider starts to transmit the carrier.	Connection Established/Transmitting
Connection Established/Transmitting	Service User invokes <i>terminateTransmit</i>		Service provider stops the transmission.	Connection Established/Receiving
Waiting for disconnection confirm	Service Provider invokes <i>disConnectInd</i>		Service User notifies waveforms of the lost connection.	Service Provider invokes <i>disConnectInd</i>

1.5 REFERENCED DOCUMENTS.

<u>Document No.</u>	<u>Document Title</u>
MSRC-5000SCA	Software Communications Architecture Specification
MSRC-5000API	Application Program Interface Supplement to the Software Communications Architecture Specification, Appendix C Generic Packet Building Block Service Definition

2 UUID.

The UUID for this API is 7cd5b880-d1d3-11d4-8cc8-00104b23b8a2.

3 SERVICES.

Refer to Table 1 for a cross-referenced listing of services and their primitives or attributes.

Table 1. Cross-Reference of Services and Primitives

Service Group	Service	Primitives or Structure Attributes
Establish Connection	connection commands	connectionReq(mode : ModeTypes) : void initiateTransmit() :void terminateTransmit() : void disconnectReq() : void
	connection Signals	connectionConfirm(connectionIds : connectionsIdType, status : boolean, dataRate : short) : void connectionInd(connectionIds : connectionsIdType, dataRate : short) : void DisconnectInd() : void
Connection Established :Transmit and Receive samples	pushPacket down stream to transmit	<i>pushPacket</i> (priority : octet, control : in HfControlType, payload : in PayloadType) : void
	pushPacket up stream for receive	<i>pushPacket</i> (priority : octet, control : in HfControlType, payload : in PayloadType) : void
	HfControlType	<i>Id</i> <i>StreamControl</i>

3.1 ESTABLISH CONNECTION.

Establish Connection provides a service that allows the establishment, maintenance, and disconnection of an ALE connection. Figure 2 provides a sequence of events that occur to establish a HF ALE connection while Figure 3 provides a sequence of events to process a incoming connection request. The following steps are illustrated in figure 2.

1. The Service User initiates a *connectionReq*(mode: ModeType). The "mode" field specifies whether the connection will be a data connect or a voice connection. This is an asynchronous event: meaning the Service Provider will notify the Service User at some future time the result of that request. The Service User will be notified via the *connectionConfirm* signal.
2. The Service Provider signals the Service User the result of the *connectionReq* via *connectionConfirm* (connectionIds, status, dataRate). The "connectionIds" field provides an array of connection IDs. When a HF ALE connection is established, more than one channel may be available on this connection. A connection ID is assigned to each channel on completion of physical connection. The "status" field specifies the result of the *connectionReq* (e.g. connectionPassed or connectionFailed). The dataRate field indicates the data rate for each of the channels.
3. When the Service User has data to send, the *initiateTransmit* is invoked to initiate the carrier.
4. Service User pushes packets downstream to the Service Provider to transmit. The "priority" field of a queue is associated with a channel. The "control" identifies the channel ID and identifies the beginning of stream. The "payload" field contains an array of 16-bit samples.
5. Service User pushes another packet downstream to the Service Provider to transmit. The "control" identifies the channel ID and identifies the end of stream.
6. When the transmission is complete and the Service User has no more data streams to transmit, *terminateTransmit* is invoked. The Service Provider ends the transmission but the connection is maintained by the Service Provider.
7. The Service Provider receives data and passes the samples to the Service User via *pushPacket*. An asynchronous reception can occur at anytime while the connection is open. *pushPacket* upstream has the same parameters as *pushPacket* downstream.
8. The Service User invokes the *disconnectReq* to close the HF ALE Connection.
9. The Service Provider signals the Service User that the connection has been disconnected. The Service User can receive a unsolicited *disconnectInd* when there is a failure in the communication path. The Service User must repeat steps 1 & 2 to receive or transmit again.

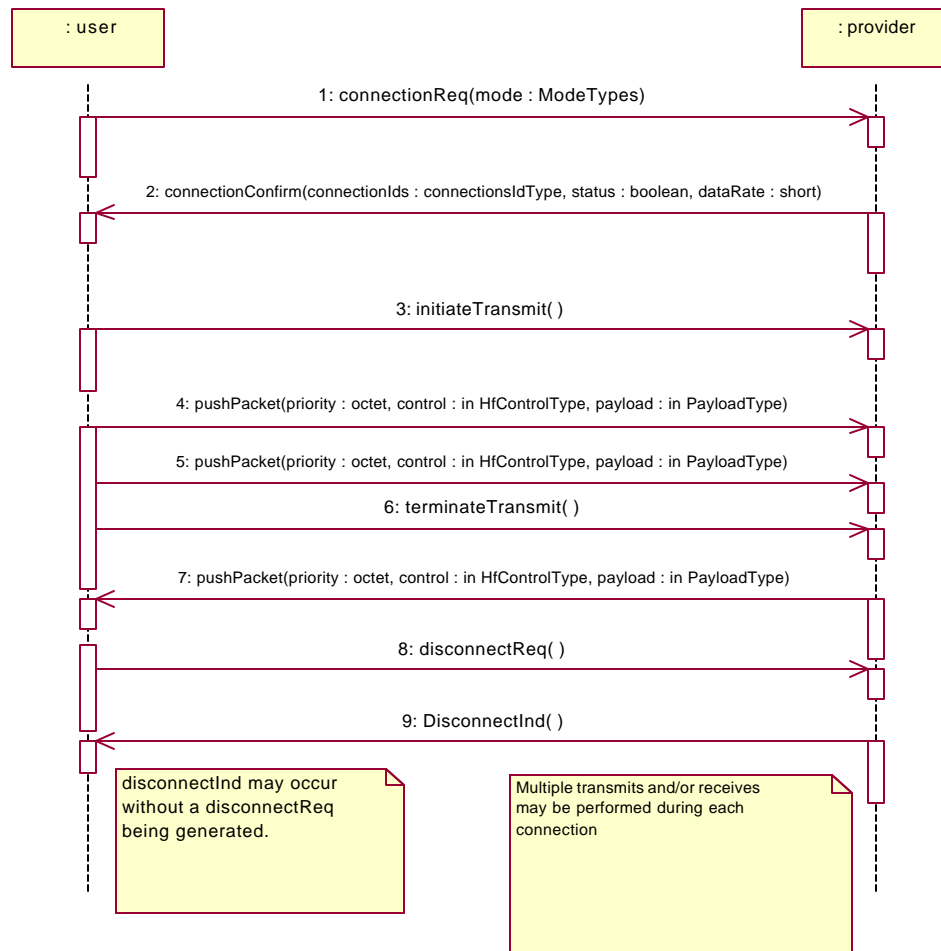


Figure 2. Sequence Diagram for Service User Local Initiated Connection

The following steps, for incoming connection, are illustrated in figure 3.

- A. Service Provider signals the Service User of a HF ALE connection from a peer via *connectionInd*(connectionIds,dataRate). The "connectionIds" field provides an array of connection IDs. When a HLE Ale connection is established, more that one channel may be available on this connection. A connection ID is assigned to each channel on physical connection. The dataRate field indicates the data rate for each of the channels.
- B. Service Provider pushes received samples upstream to the Service Provider. The "priority" field of a queue is associated with a channel. The "control" identifies the channel ID and identifies the beginning of stream. The "payload" field contains an array of 16-bit samples.
- C. When the Service User has data to send, the *initiateTransmit* is invoked to initiate the carrier.
- D. Service User pushes samples downstream to the Service Provider. The "priority" field of a queue is associated with a channel. The "control" identifies the channel ID and identifies

the beginning of stream and end of stream (complete transmission in payload of this *pushpacket*). The "payload" field contains an array of 16-bit samples.

- E. When the transmission is complete and the Service User has no more data streams to transmit, *terminateTransmit* is invoked. The Service Provider ends the transmission but the connection is maintained by the Service Provider.

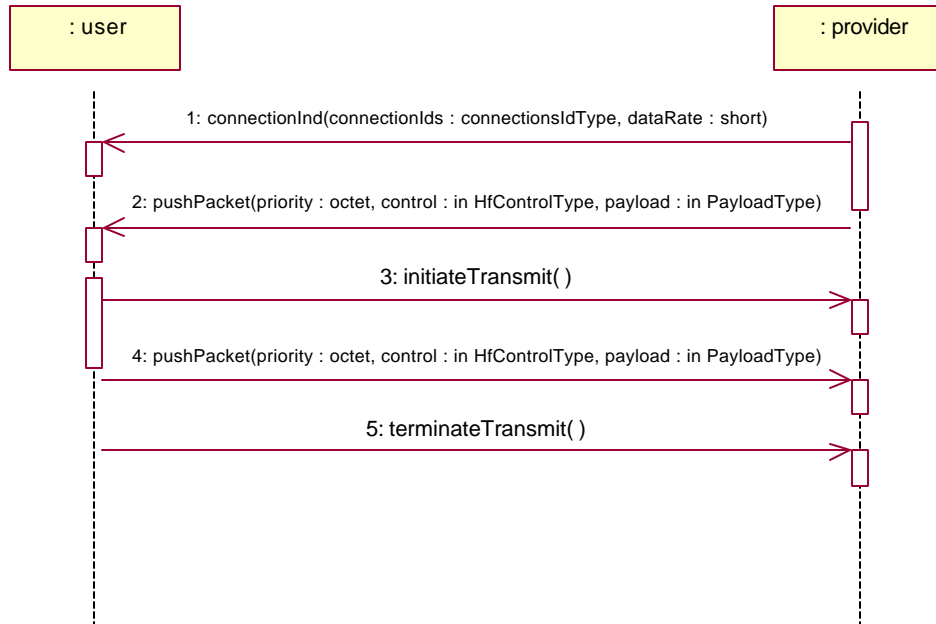


Figure 3. Incoming Connection Sequence Diagram

3.2 TRANSMIT AND RECEIVE SAMPLES.

*FDM

4 SERVICE PRIMITIVES.

4.1 CONNECTIONCOMMANDS.

This interface (see Figure 4) provides methods associated with a HF ALE connection for the Service Provider to invoke and the Service Provider to implement.

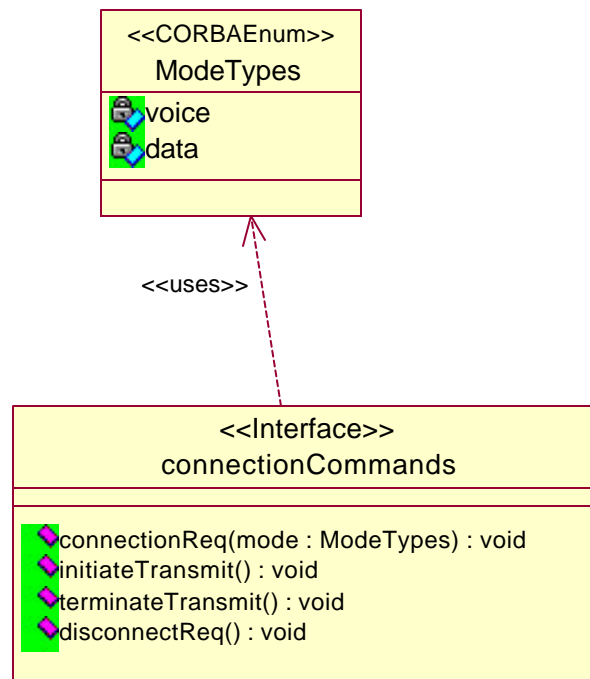


Figure 4. connectionCommands

4.1.1 connectionReq.

Upon receiving a **connectionReq**, the ALE system attempts to perform a data handshake with another station using the best available frequency. Once the two-way link is established, normal voice or data communication can begin.

4.1.1.1 Synopsis.

connectionReq(mode : ModeTypes) : void

4.1.1.2 Parameters.

mode : ModeTypes

This parameter indicates which mode (i.e. data or voice) to invoke the connection State.

4.1.1.3 State.

No connection

4.1.1.4 New State.

Waiting for connection Confirm.

4.1.1.5 Response.

None.

4.1.1.6 Originator.

Service User

4.1.1.7 Errors/Exceptions.

None.

4.1.2 initiateTransmit.

When in a link, the HF ALE tells the physical layer to enter the transmit mode. If the link is a voice link, HF ALE resets an internal link keep-alive timer.

4.1.2.1 Synopsis.

initiateTransmit() : void

4.1.2.2 Parameters.

None.

4.1.2.3 State.

Connection Established

4.1.2.4 New State.

Transmitting

4.1.2.5 Response.

None.

4.1.2.6 Originator.

Service User

4.1.2.7 Errors/Exceptions.

None.

4.1.3 terminateTransmit.

The HF ALE tells the physical layer to exit the transmit mode, and HF ALE returns to receive monitoring for HF ALE transmissions.

4.1.3.1 Synopsis.

terminateTransmit() : void

4.1.3.2 Parameters.

None.

4.1.3.3 State.

Transmitting.

4.1.3.4 New State.

Receiving.

4.1.3.5 Response.

None.

4.1.3.6 Originator.

Service User.

4.1.3.7 Errors/Exceptions.

None.

4.1.4 disconnectReq.

The HF ALE transmits a link termination to the other station, then returns the system to receive scanning and monitors for HF ALE transmissions.

4.1.4.1 Synopsis.

disconnectReq() : void

4.1.4.2 Parameters.

None.

4.1.4.3 State.

Connection Established.

4.1.4.4 New State.

No Connection .

4.1.4.5 Response.

None.

4.1.4.6 Originator.

Service User.

4.1.4.7 Errors/Exceptions.

None.

4.2 CONNECTIONSIGNALS.

This interface (Figure 5) provides methods for the Service Provider to notify the Service Provider of asynchronous connection events. The Service Provider will invoke the methods and the Service User will implement the methods.

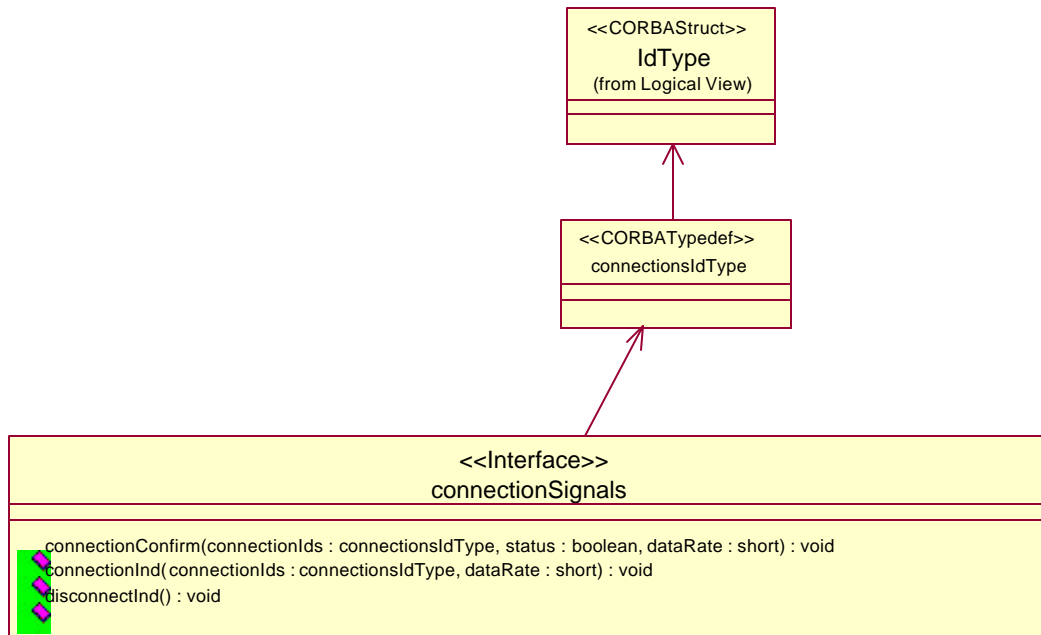


Figure 5. Connection Signals

4.2.1 connection Confirm.

This method is called when an HF ALE *connectRequest* operation is complete. If status is false, the link attempt has failed. If status is true, connectionIds indicates the RF communication channels that are available, and dataRate indicates the maximum data rate suggested by the link quality.

4.2.1.1 Synopsis.

`connectionConfirm(connectionIds : connectionsIdType, status : boolean, dataRate : short) : void`

4.2.1.2 Parameters.

4.2.1.2.1 connectionIds.

connectionIds is sequence of octets. Each octet identifies a channel. "connctionId" may be queried for its size which indicates the number of channels available on this connection. If the connection fails, the size of this array shall be 0.

4.2.1.2.2 status

True when a connection was established, otherwise false.

4.2.1.2.3 dataRate

Indicates the maximum data rate recommended for each channel.

4.2.1.3 State.

No connection.

4.2.1.4 New State.

Connection established.

4.2.1.5 Response.

None.

4.2.1.6 Originator.

Service Provider

4.2.1.7 Errors/Exceptions.

None.

4.2.2 connectionInd.

This method is called when a received HF ALE link is complete. connectionIds indicates the RF communication channels that are available, and dataRate indicates the maximum data rate suggested by the link quality.

4.2.2.1 Synopsis.

connectionInd(connectionIds : connectionsIdType, dataRate : short) : void

4.2.2.2 Parameters.

4.2.2.2.1 connectionIds.

connectionIds is sequence of octets. Each octet identifies a channel. connectionIds may be queried for its size which indicates the number of channels available on this connection. If the connection fails the size of this array shall be 0.

4.2.2.2.2 dataRate.

Indicates the maximum data rate recommended for each channel in samples per second: a sample is 16 bits.

4.2.2.3 State.

No connection.

4.2.2.4 New State.

Connection established.

4.2.2.5 Response.

None.

4.2.2.6 Originator.

Service Provider.

4.2.2.7 Errors/Exceptions.

None.

4.2.3 disconnectInd.

Indicates the HF ALE link with the other station has been terminated. HF ALE returns the system to receive scanning and monitors for HF ALE transmissions.

4.2.3.1 Synopsis.

disconnectInd() : void

4.2.3.2 Parameters.

None.

4.2.3.3 State.

Connection Established.

4.2.3.4 New State.

No Connection.

4.2.3.5 Response.

None.

4.2.3.6 Originator.

Service Provider.

4.2.3.7 Errors/Exceptions.

None.

4.3 CONNECTION ESTABLISHED : TRANSMIT AND RECEIVE SAMPLES.

This interface provides methods for the Service User to send samples to the Service Provider to be transmitted, and provides methods for the Service Provider to send received samples to the Service User.

4.4 MACPUSHPACKET.

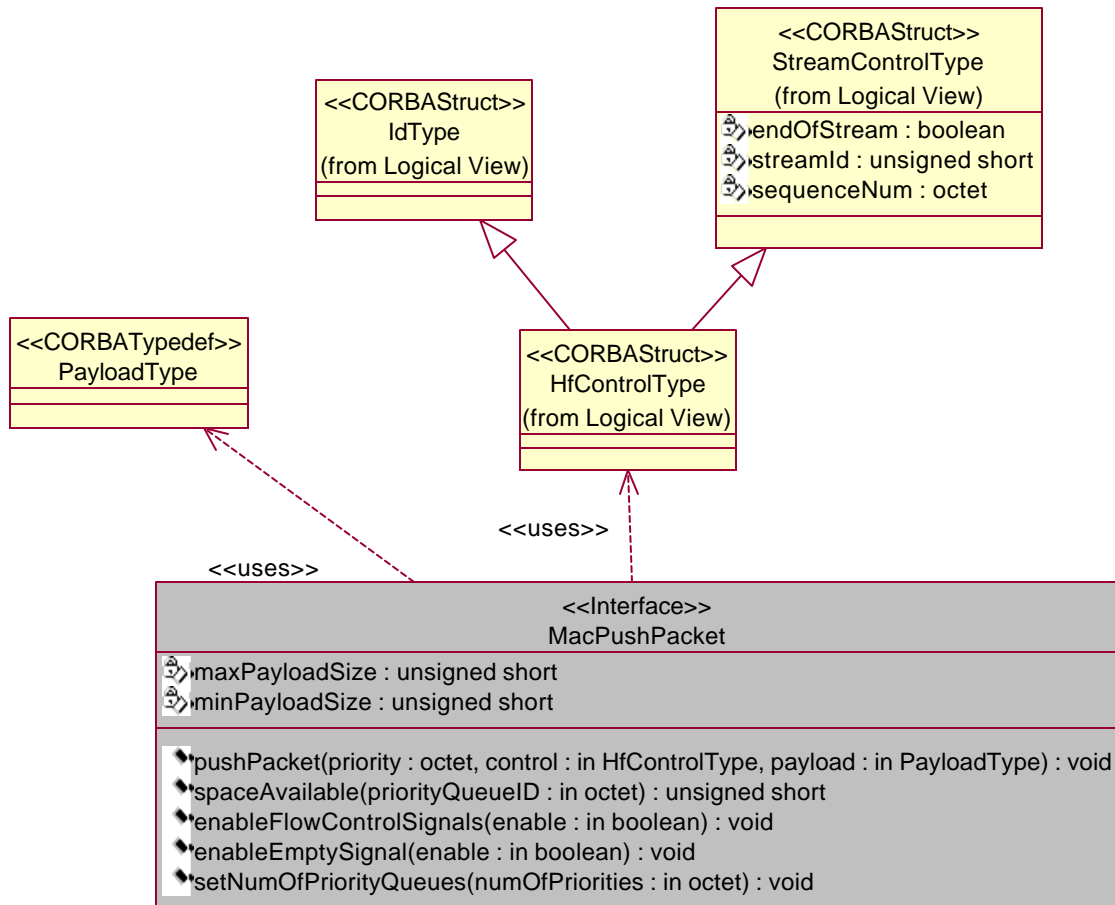


Figure 6. Packet BB

4.4.1 spaceAvailable.

This operation provides the ability to poll the Service Provider to determine the amount of space available in samples (16-bit samples) in the queue for the given priority. The Service User will poll the Server Provider to determine how much room is available in samples in the specified priority. When the operation is invoked, the server will respond with the amount of available space on the queue, in samples.

4.4.1.1 Synopsis.

`spaceAvailable(priorityQueueID : in octet): return short`

4.4.1.2 Parameters.

`priorityQueueID : octet`

This parameter indicates which `PriorityQueue` to check. The number of priority queues is set up via `SetNumOfPriorityQueues` primitive. If `SetNumOfPriorityQueues` has not been called, the default number of priority queues is 1.

4.4.1.3 State.

Any state.

4.4.1.4 New State.

Same state.

4.4.1.5 Response.

This operation responds with the amount of available space on the specified queue in samples.

4.4.1.6 Originator.

Service User.

4.4.1.7 Errors/Exceptions.

QUEUE_NOT_DEFINED

4.4.2 enableFlowControlSignals.

This operation is used to activate and deactivate the ‘water-mark’ signals. The default is false (signals will not be generated).

4.4.2.1 Synopsis.

enableFlowControlSignals(enable : in boolean) : void

4.4.2.2 Parameters.

enable: boolean

false: The Service Provider will not generate signals to indicate the Lowwater and Highwater queue conditions. It is up to the Service User to poll the Service Provider to insure the Service Provider will not be starved or the queue will not overflow. The instantiating API should define behavior upon starvation or queue overflow.

true: The Service Provider will signal the Lowwater and Highwater queue conditions, to the Service User, when the Lowwater has been reached (queue near empty).

4.4.2.3 State.

NO_PROVIDER_WATERMARK_SIGNALS or PROVIDER_WATERMARK_SIGNALS

4.4.2.4 New State.

True -> PROVIDER_WATERMARK_SIGNALS

False-> NO_PROVIDER_WATERMARK_SIGNALS

4.4.2.5 Response.

None.

4.4.2.6 Originator.

Service User.

4.4.2.7 Errors/Exceptions.

None.

4.4.3 enableEmptySignal.

This operation is used to activate and deactivate the 'empty' signal. The signal will not be generated when set to False.

4.4.3.1 Synopsis.

enableEmptySignal(enable : in boolean) : void

4.4.3.2 Parameters.

enable : boolean

false: The Service Provider will not generate a signal to indicate all queues are empty. It is up to the Service User to poll the Service Provider to insure the Service Provider will not be starved. The instantiating API should define behavior upon starvation.

true: The Service Provider will generate a signal to the Service User when the all queues are empty.

4.4.3.3 State.

NO_PROVIDER_EMPTY_SIGNAL or PROVIDER_EMPTY_SIGNAL

4.4.3.4 New State.

mode(on) -> PROVIDER_EMPTY_SIGNAL

mode(off)-> NO_PROVIDER_EMPTY_SIGNAL

4.4.3.5 Response.

None.

4.4.3.6 Originator.

Service User.

4.4.3.7 Errors/Exceptions.

None.

4.4.4 setNumOfPriorityQueues.

This operation is used by the Service User to inform the Service Provider how many priority queues to provide.

4.4.4.1 Synopsis.

setNumOfPriorityQueues(numOfPriorities : in octet) : void

4.4.4.2 Parameters.

numOfPriorities : octet Specifies the number of priorities the Service Provider should process. (e.g., If the Service Provider set the value to 10, the Service User would send packets to the Service Provider with a priority of 0-9 (where 9 is the highest priority). Messages in priority 9 will be processed first by the Service Provider.)

4.4.4.3 State.

Any state.

4.4.4.4 New State.

Same state.

4.4.4.5 Response.

None.

4.4.4.6 Originator.

Service User.

4.4.4.7 Errors/Exceptions.

EXCEEDS_CAPACITY

4.4.5 getMaxPayLoadSize.

Returns the maxPayLoadSize in samples. This operation is auto-generated from the associated attribute.

4.4.5.1 Synopsis.

getMaxPayLoadSize(void) : unsigned short

4.4.5.2 Parameters.

None.

4.4.5.3 State.

Any state.

4.4.5.4 New State.

Same state.

4.4.5.5 Response.

Returns the maxPayLoadSize in samples.

4.4.5.6 Originator.

Service User.

4.4.5.7 Errors/Exceptions.

None.

4.4.6 getMinPayLoadSize.

Returns the minPayLoadSize in samples. This operation is auto-generated from the associated attribute.

4.4.6.1 Synopsis.

getMinPayLoadSize(void) : unsigned short

4.4.6.2 Parameters.

None.

4.4.6.3 State.

Any state.

4.4.6.4 New State.

Same state.

4.4.6.5 Response.

This operation returns the minPayLoadSize in samples.

4.4.6.6 Originator.

Service User.

4.4.6.7 Errors/Exceptions.

None.

4.4.7 pushPacket.

"pushPacket" provides the ability of pushing data packets from the Service User to a Service Provider and from the Service Provider to the Service User. A packet is made up of two parts control and payload. The payload is queued according to the priority and is processed according to the information specified in control parameter.

4.4.7.1 Synopsis.

pushPacket(priority : octet, control : in HfControlType, payload : in PayloadType) : void

4.4.7.2 Parameters.

priority: octet The priority queue to put the control and associated payload in. (See setNumOfPriorityQueues)

control: HfControlType

payload: sequence of short;

4.4.7.3 State.

connected

4.4.7.4 New State.

connected

4.4.7.5 Response.

None.

4.4.7.6 Originator.

Service User or Service Provider.

4.4.7.7 Errors/Exceptions.

None.

4.5 SIGNALS BB.

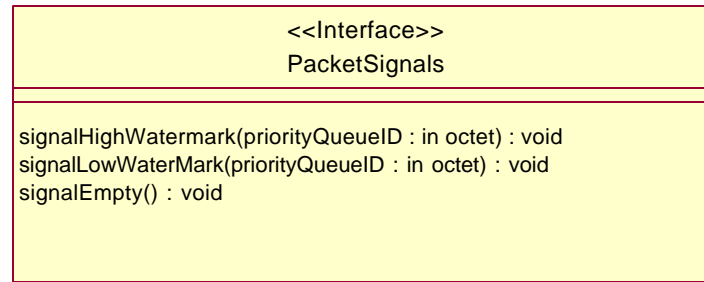


Figure 7. Signal BB

4.5.1 signalHighWatermark.

signalHighWaterMark provides the ability to signal the Service User when a queue has reached the high water mark: the queue is full for the specified priority.

4.5.1.1 Synopsis.

signalHighWatermark(priorityQueueID : in octet) : void

4.5.1.2 Parameters.

priorityQueueID : octet indicates the queue priority which has reached the high water mark. (See setNumOfPriorityQueues).

4.5.1.3 State.

Any state.

4.5.1.4 New State.

Same state.

4.5.1.5 Originator.

Service Provider.

4.5.1.6 Errors/Exceptions.

None.

4.5.2 signalLowWatermark.

signalLowWaterMark provides the ability to signal the service user when a queue has reached the low water mark: the queue is near empty for the specified priority.

4.5.2.1 Synopsis.

signalLowWatermark(priorityQueueID : in octet) : void

4.5.2.2 Parameters.

priorityQueueID : octet indicates the queue priority which has reached the low water mark. (See setNumOfPriorityQueues).

4.5.2.3 State.

Any state.

4.5.2.4 New State.

Same state.

4.5.2.5 Originator.

Service Provider.

4.5.2.6 Errors/Exceptions.

None.

4.5.3 signal Empty.

signalEmpty provides the ability to signal the Service User when all priority queues are empty. (See *setNumOfPriorityQueues*).

4.5.3.1 Synopsis.

signalEmpty(void) : void

4.5.3.2 Parameters.

None.

4.5.3.3 State.

Any state.

4.5.3.4 New State.

Same state.

4.5.3.5 Originator.

Service Provider.

4.5.3.6 Errors/Exceptions.

None.

4.6 COMMON STRUCTURES

The follow structures are used to create a *Control_Type*.

4.6.1 Stream Control Structure.

Stream control structure is used to control data groups sent between Service User and Service Provider.

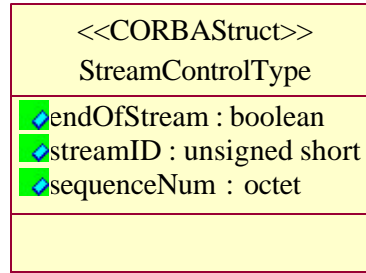


Figure 8. Stream Control

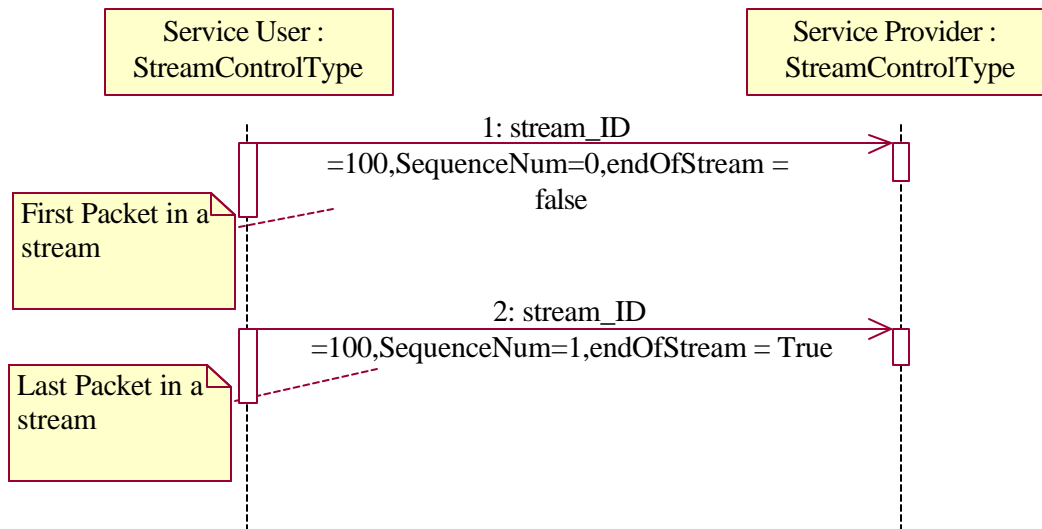


Figure 9. Stream Control Sequence Diagram

4.6.1.1 endOfStream.

Indicates last group of symbols for this hop: end of stream.

4.6.1.2 steamId.

Identifies the groups of symbols to be transmitted or received in one hop.

4.6.1.3 sequenceNum.

Sequence number of the group of symbols within the stream sequence. The waveform application sets this value to zero at the start of stream. If value is set to zero, it indicates beginning of stream.

5 ALLOWABLE SEQUENCE OF SERVICE PRIMITIVES.

Table 2. High Water and Low Water and Empty On

Current State	Logical Event	Condition	Action	Next State
Queue normal	<i>PushPacket</i>	<i>Pushpacket</i> does not cause queue to reach high water mark for the specified priority.	queue packet	Queue normal
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet <i>signalHighWatermark.</i>	Queue at high water mark
		Pushpacket has Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	No state change
	packet extracted from queue	Packet extracted causes low water mark to be reached	<i>signalLowWatermark</i>	Queue at low water mark
		Packet extracted causes low water mark not to be reached		Queue normal
		Packet extracted causes all queues to be empty.	<i>signalEmpty.</i>	Queue empty
Queue at low water	<i>PushPacket</i>	<i>Pushpacket</i> does not cause the queue to exceed the low water mark for the specified priority.	queue packet	Queue at low water
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet <i>signalHighWatermark</i>	Queue at high water mark
		<i>Pushpacket</i> causes the queue to exceed the low water mark for the specified priority but less than the high water	queue packet	Queue normal

		mark.		
		Pushpacket has a Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes all queues to be empty	<i>signalEmpty</i>	Queue empty
		Packet extracted does not causes all queues to be empty		Queue at low water
Queue at high water	<i>PushPacket</i>		*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes low water mark to be reached	<i>signalLowWatermark</i>	Queue at low water mark
		Packet extracted causes all queues to be empty	<i>signalEmpty</i>	Queue empty
		Packet extracted causes the specified queue to be less than high water and greater than low water		Queue normal
		Queue is still greater than or equal to high water mark		Queue at high water
Queue empty	<i>PushPacket</i>	<i>Pushpacket</i> causes the queue to be greater than the low water mark for the specified priority but less than the high water mark	queue packet	Queue normal
		<i>Pushpacket</i> causes the queue to equal the low water mark for the specified priority	<i>signalLowWatermark</i>	Queue at low water mark
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet <i>signalHighWatermark</i>	Queue at high water mark

		Pushpacket has a Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state.
	attempt to extract packet from queue	.	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state.

Table 3. High Water and Low Water Off and Empty On

Current State	Logical Event	Condition	Action	Next State
Queue normal	<i>PushPacket</i>	<i>Pushpacket</i> does not cause queue to reach high water mark for the specified priority	queue packet	Queue normal
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet	Queue at high water mark
		<i>Pushpacket</i> has Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes low water mark to be reached		Queue at low water mark
		Packet extracted causes low water mark not to be reached		Queue normal
		Packet extracted causes all queues to be empty	<i>signalEmpty.</i>	Queue empty
Queue at low water.	<i>PushPacket</i>	<i>Pushpacket</i> does not cause the queue to exceed the low water mark for the specified priority	queue packet	Queue at low water
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet	Queue at high water mark
		<i>Pushpacket</i> causes the queue to exceed the low water mark for the specified priority but less than the high water mark	queue packet	Queue normal

		Pushpacket has a Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes all queues to be empty	<i>signalEmpty</i> .	Queue empty
		Packet extracted does not causes all queues to be empty		Queue at low water
Queue at high water	<i>PushPacket</i>		*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes low water mark to be reached		Queue at low water mark
		Packet extracted causes all queues to be empty.	<i>signalEmpty</i> .	Queue empty
		Packet extracted causes the specified queue to be less than high water and greater than low water		Queue normal
		Queue is still greater than or equal to high water mark		Queue at high water
Queue empty	<i>PushPacket</i>	<i>Pushpacket</i> causes the queue to be greater than the low water mark for the specified priority but less than the high water mark	queue packet	Queue normal
		<i>Pushpacket</i> causes the queue to equal the low water mark for the specified priority		Queue at low water mark
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet	Queue at high water mark
		Pushpacket has a Payload larger than can	*Attempt to put in next lower queue else	Same state

		be put into the queue	raise exception PAYLOAD_TO_BIG	
	attempt to extract packet from queue		Stop Transmission	Same state

Table 4. High Water and Low Water and Empty Off

Current State	Logical Event	Condition	Action	Next State
Queue normal	<i>PushPacket</i>	<i>Pushpacket</i> does not cause queue to reach high water mark for the specified priority	queue packet	Queue normal
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet	Queue at high water mark
		<i>Pushpacket</i> has Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state.
	packet extracted from queue	Packet extracted causes low water mark to be reached		Queue at low water mark
		Packet extracted causes low water mark not to be reached		Queue normal
		Packet extracted causes all queues to be empty		Queue empty
Queue at low water	<i>PushPacket</i>	<i>Pushpacket</i> does not cause the queue to exceed the low water mark for the specified priority	queue packet	Queue at low water
		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the	queue packet	Queue at high water mark

		<i>Payload</i>		
		<i>Pushpacket</i> causes the queue to exceed the low water mark for the specified priority but less than the high water mark	queue packet	Queue normal
		<i>Pushpacket</i> has a Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes all queues to be empty		Queue empty
		Packet extracted does not causes all queues to be empty		Queue at low water
Queue at high water	<i>PushPacket</i>		*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	packet extracted from queue	Packet extracted causes low water mark to be reached		Queue at low water mark
		Packet extracted causes all queues to be empty		Queue empty
		Packet extracted causes the specified queue to be less than high water and greater than low water		Queue normal
		Queue is still greater than or equal to high water mark		Queue at high water
Queue empty	<i>PushPacket</i>	<i>Pushpacket</i> causes the queue to be greater than the low water mark for the specified priority but less than the high water mark	queue packet	Queue normal
		<i>Pushpacket</i> causes the queue to equal the low water mark for the specified priority		Queue at low water mark

		<i>Pushpacket</i> causes queue to reach high water mark for the specified priority and there is room in the queue to put the <i>Payload</i>	queue packet	Queue at high water mark
		Pushpacket has a Payload larger than can be put into the queue	*Attempt to put in next lower queue else raise exception PAYLOAD_TO_BIG	Same state
	attempt to extract packet from queue	.	Stop transmission	Same state

6 PRECEDENCE OF SERVICE PRIMITIVES.

This section is intentionally blank.

7 SERVICE USER GUIDELINES.

This section is intentionally blank.

8 SERVICE PROVIDER-SPECIFIC INFORMATION.

Not applicable.

9 IDL.

The HF ALE MAC interface design depicted in IDL source code is shown in the following subsections.

9.1 IDL FOR ALE CONFIGURE.

```
//Source file: C:/Projects/JTRS/APIs/SCAWorkingGroup/Building_Blocks/HF-  
ALE/Mac_API/NRTidl/ALEConfigure.idl
```

```
#ifndef __ALECONFIGURE_DEFINED  
#define __ALECONFIGURE_DEFINED
```

```
/* CmIdentification  
   %X% %Q% %Z% %W% */
```

```
#include "ALEResponses.idl"
```

```
struct lqaParametersType {  
    callAddressType lqaCallAddress;  
    unsigned frequencyInHz;  
    unsigned short lqaTimeInSeconds;  
    unsigned short rxLQABER;  
    unsigned short rxLQASINAD;  
    unsigned short txLQABER;  
    unsigned short txLQASINAD;  
};
```

```
typedef string <15> callAddressType;
```

```
struct systemParametersType {  
    boolean AllCall;  
    boolean AMDInACK;  
    boolean callAlert;  
    boolean commandLQA;  
    boolean lbcEnable;  
    boolean terminateLinkTransmission;  
    unsigned short keepAliveTxIntervalInSecs;  
    unsigned short listenBeforeCallTimeInMs;  
    unsigned short lqaDegradeIntervalInMinutes;  
    unsigned short networkTuneTimeInSecs;  
    unsigned short returnToScanTimeInSecs;
```

```

        unsigned short minScanDwellTimeInMs;
        unsigned short staticModeCallSoundDurationInSecs;
};

struct starGroupParametersType {
    unsigned short index;
    any groupNameCallAddressType;
    unsigned short numberOfMembers;
    sequence <callAddressType,8> membersList;
};

struct scanListParametersType {
    unsigned scanListNumber;
    boolean defaultToQuickALE;
    boolean otherCallProtocalls;
    unsigned short callDurationInseconds;
    boolean enableSounding;
    unsigned short soundDurationInseconds;
    unsigned short numberOfScanChannels;
    sequence <unsigned,20> channelsToScan;
};

struct starNetParametersType {
    boolean respondentsActive;
    boolean fixedLengthAddress;
    unsigned short numberOfRespondents;
    sequence <callAddressType, 24> respondentList;
    unsigned short index;
    callAddressType starNetName;
    boolean useScanLists;
    unsigned short scanListIndex;
};

struct channelParametersType {
    unsigned short channelNumber;
    unsigned txFreqInHz;
    emissionModeType txEmissionMode;
    unsigned short txPowerLevel;
    unsigned rxFreqInHz;
    emissionModeType rxEmissionMode;
    unsigned short soundingIntervalInMinutes;
    boolean rxOnly;
    boolean enableSounding;
};

enum callProtocolType {
    Default,
    StandardALE,
    QuickALE
};

struct addressParametersType {
    unsigned short addressIndex;
    callAddressType otherAddress;
    unsigned short scanListIndex;
    unsigned short remoteStationTuneTimeInSeconds;
    callProtocolType callProtocol;
};

```

```

};

enum emissionModeType {
    USB,
    LSB,
    UUSB,
    LLSB
};

enum scanListModeType {
    None,
    AllLists,
    SelectedList
};

struct selfAddressParametersType {
    boolean useNets;
    unsigned short netAddressIndex;
    boolean netResponseActive;
    unsigned short index;
    callAddressType selfAddress;
    scanListType scanListMode;
    unsigned short scanListIndex;
};

interface ConfigureALE {
    /*
    @roseuid 3A131CC80078 */
    boolean configureChannel (
        in channelParametersType ChannelParameters
    );

    /*
    @roseuid 3A14435C0156 */
    channelParametersType ListChannel (
        in unsigned short ChannelNumber
    );

    /*
    @roseuid 3A131D2B00CB */
    boolean configureScanList (
        in scanListParametersType ScanListParameters
    );

    /*
    @roseuid 3A131D6F023B */
    boolean configureOtherAddress (
        in addressParametersType AddressParameters
    );

    /*
    @roseuid 3A131E4C01A3 */
    boolean configureSelfAddress (
        in selfAddressParametersType SelfAddressParameters
    );

    /*

```

```

@roseuid 3A131E830300 */
boolean configureStarGroup (
    in starGroupParametersType StarGroupParameters
);

/*
@roseuid 3A131EAA01DA */
boolean configureStarNet (
    in starNetParametersType StartNetParameters
);

/*
@roseuid 3A131EFD0071 */
boolean configureLQA (
    in lqaParametersType LQAParameters
);

/*
@roseuid 3A14444803C2 */
boolean listKey (
    in unsigned short keyNumber,
    in lpLevelType1Type keyType
);

/*
@roseuid 3A131FD8036F */
boolean configureSystemParameters (
    in systemParametersType SystemParameters
);

/*
@roseuid 3A14665200C1 */
boolean zeroizeData ();

};

#endif

```

9.2 IDL FOR ALE OPERATIONS.

//Source file: C:/Projects/JTRS/APIs/SCAWorkingGroup/Building_Blocks/HF-ALE/Mac_API/NRTidl/ALEOperations.idl

```

#ifndef __ALEOPERATIONS_DEFINED
#define __ALEOPERATIONS_DEFINED

```

```

/* CmIdentification
   %X% %Q% %Z% %W% */

```

```

#include "ALEResponses.idl"

```

```

enum callTypeType {
    Unknown,
    individual,
    Net,
    All,
}

```

```

    Any,
    Group,
    Sound,
    None
};

struct qaListParametersType {
    TimeType lqaTime;
    unsigned FrequencyInHz;
    callAddressType lqaAddress;
    unsigned short minimumLQAValue;
    unsigned short commandNumber;
};

typedef string <15> callAddressType;

struct soundCommandParametersType {
    boolean soundAll;
    unsigned short channelToSound;
    boolean soundImmediate;
    unsigned short soundCommandNumber;
};

struct systemStatusResponseType {
    systemModeType systemMode;
    unsigned short scanIndex;
    boolean Tx;
    aleStateType alestate;
    unsigned short channelNumber;
    callTypeType callType;
    callAddressType thierAddress;
    emissionModeType txEmissionMode;
    unsigned TxFrequencyInHz;
    emissionModeType rxEmissionMode;
    callAddressType lastAddress;
    unsigned rxFrequencyInHz;
    callAddressType myAddress;
};

enum systemModeType {
    Standby,
    Datafill,
    ALEScan,
    ALEPreset,
    ALEManual,
    Test
};

struct CallParametersType {
    unsigned short callNumber;
    callAddressType callAddress;
    unsigned short scanListIndex;
    boolean noLinkCommand;
    boolean useAMD;
    unsigned short amdCommand;
    boolean lqaCommand;
    callTypeType callType;
};

```

```

};

interface ALEOperations {
    /*
    @roseuid 3A14484D02D1 */
    void SetSystemMode (
        in unsigned short CommandID,
        in systemModeType SystemMode,
        in unsigned short Index
    );

    /*
    @roseuid 3A1449FA0120 */
    boolean setCallAddress (
        in callAddressType address
    );

    /*
    @roseuid 3A144A20023D */
    boolean setScanList (
        in unsigned short NumberOfScanLists,
        in sequence <unsigned short ScanLists,
        any 20>
    );

    /*
    @roseuid 3A144A5D0230 */
    void call (
        in callParametersType CallParameters
    );

    /*
    @roseuid 3A144CA00084 */
    void soundCommand (
        in soundParametersType soundParameters
    );

    /*
    @roseuid 3A144FA90096 */
    void listQA (
        in qaListParametersType QAListParameters
    );
};

#endif

```

9.3 IDL FOR ALE RESPONSES.

//Source file: C:/Projects/JTRS/APIs/SCAWorkingGroup/Building_Blocks/HF-ALE/Mac_API/NRTidl/ALEResponses.idl

```

#ifndef __ALERESPONSES_DEFINED
#define __ALERESPONSES_DEFINED

```

```

/* CmIdentification

```

```

%X% %Q% %Z% %W% */

#include "ALEOperations.idl"

enum callTypeType {
    Unknown,
    individual,
    Net,
    Sound,
    Group,
    Any,
    None,
    All
};

typedef string <15> callAddressType;

enum aleStateType {
    Answering,
    CallFailed,
    CallInProgress,
    Linked,
    Sounding,
    CallFailedOnAll,
    StoppedScanning,
    Calling,
    Listening
};

enum sytemModeType {
    Standby,
    Test,
    ALEManual,
    ALEPreset,
    ALEScan,
    Datafill
};

struct CallEventResponseType {
    callTypeType callType;
    callAddressType ourAddress;
    callAddressType thierAddress;
    callAddressType linkedMasterAddress;
    unsigned short channelNumber;
    unsigend short channelRanking;
    callEventType typeOfCallEvent;
    boolean lastResponse;
    unsigned short commandNumber;
};

enum emissionModeType {
    USB,
    LSB,
    UUSB,
    LLSB
};

```

```

struct systemStatusResponseType {
    callTypeType callType;
    callAddressType myAddress;
    callAddressType lastAddress;
    unsigned TxFrequencyInHz;
    emissionModeType txEmissionMode;
    unsigned rxFrequencyInHz;
    emissionModeType rxEmissionMode;
    boolean Tx;
    callAddressType thierAddress;
    unsigned short scanIndex;
    unsigned short channelNumber;
    systemModeType systemMode;
    aleStateType alestate;
};

enum systemModeType {
    Standby,
    Test,
    Datafill,
    ALEScan,
    ALEPreset,
    ALEManual
};

interface ALEResponses {
    /*
    @roseuid 3A14528703D0 */
    void callEventInfoResponse (
        in callEventResponseParametersType CallEventResponseParameters
    );

    /*
    @roseuid 3A14530000B3 */
    void commandAckResponse (
        in unsigned short CommandNumber,
        in boolean Success
    );

    /*
    @roseuid 3A14536101DF */
    void systemModeResponse (
        unsigned short CommandNumber,
        systemModeType SystemMode,
        boolean SystemGo,
        boolean SystemOperational,
        boolean InhibitTx
    );

    /*
    @roseuid 3A1454120111 */
    void amdResponse (
        in timeType TimeOfCommand,
        callAddressType Source,
        string<90> Message
    );
};

```

```

/*
@roseuid 3A14547F01A4 */
void ReceivedRespondeesResponse (
    callAddressType RespondeeAddress
);

/*
@roseuid 3A1454F30273 */
void listLQAResponse (
    lqaResponseParametersType LQAResponseParameters
);

/*
@roseuid 3A146E6A00B6 */
void systemStatusResponse (
    unsigned short commandNumber,
    boolean astResponse
);

};

#endif

```

9.4 IDL FOR CONNECTION.

//Source file: C:/Projects/JTRS/APIs/SCAWorkingGroup/Building_Blocks/HF-ALE/Mac_API/RTIDL/Connection.idl

```

#ifndef __CONNECTION_DEFINED
#define __CONNECTION_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "HF_Packet.idl"

struct IdType {
};

enum ModeTypes {
    data,
    voice
};

interface connectionCommands {
    /*
    @roseuid 39E738A60394 */
    void connectionReq (
        ModeTypes mode
    );

    /*
    @roseuid 39E742AB01BB */
    void initiateTransmit ();

    /*

```

```

    @roseuid 39E742B60009 */
    void terminateTransmit ();

    /*
    @roseuid 39E7454B024E */
    void disconnectReq ();

};

typedef sequence <IdType> connectionsIdType;

interface connectionSignals {
    /*
    @roseuid 39E73A5E0152 */
    void connectionConfirm (
        connectionsIdType connectionIds,
        boolean status,
        short dataRate
    );

    /*
    @roseuid 39E741A503BD */
    void connectionInd (
        connectionsIdType connectionIds,
        short dataRate
    );

    /*
    @roseuid 39E7427500F5 */
    void DisconnectInd ();

};

#endif

```

9.5 IDL FOR HF PACKET.

//Source file: C:/Projects/JTRS/APIs/SCAWorkingGroup/Building_Blocks/HF-ALE/Mac_API/RTIDL/HF_Packet.idl

```

#ifndef __HF_PACKET_DEFINED
#define __HF_PACKET_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

struct IdType {
};

struct StreamControlType {
    boolean endOfStream;
    unsigned short streamId;
    octet sequenceNum;
};

typedef sequence <short> PayloadType;

```

```

struct HfControlType {
};

interface MacPushPacket {
    /* The maxPacketSize is a read only attribute set by the Packet Server
    and the get operation reports back the maximum number of traffic units
    allowed in one pushPacket call.  */

    attribute unsigned short maxPayloadSize;
    attribute unsigned short minPayloadSize;

    /* This operation is used to push Client data to the Server with a
    Control element and a Payload element.
    @roseuid 39E734B1021C */
    void pushPacket (
        octet priority,
        in HfControlType control,
        in PayloadType payload
    );

    /* The operation returns the space available in the Servers queue(s) in
    terms of the implementers defined Traffic Units.
    @roseuid 39E734B10232 */
    unsigned short spaceAvailable (
        in octet priorityQueueID
    );

    /* This operation allows the client to turn the High Watermark Signal
    ON and OFF.
    @roseuid 39E734B10234 */
    void enableFlowControlSignals (
        in boolean enable
    );

    /* This operation allows the client to turn theEmpty Signal ON and OFF.
    @roseuid 39E734B10239 */
    void enableEmptySignal (
        in boolean enable
    );

    /*
    @roseuid 39E734B1023B */
    void setNumOfPriorityQueues (
        in octet numOfPriorities
    );
};

#endif

```

9.6 IDL FOR HF API.

//Source file: C:/Projects/JTRS/APIs/SCAWorkingGroup/Building_Blocks/HF-ALE/Mac_API/RTIDL/HFAPI.idl

```

#ifndef __HFAPI_DEFINED
#define __HFAPI_DEFINED

/* CmIdentification
   %X% %Q% %Z% %W% */

#include "HF_Packet.idl"
#include "Connection.idl"

interface PacketSignals {
    /* This operation is a call event back to the PacketAPI client
    indicating that a queue has reach the high watermark.  If priority or
    multiple queues are being supported then the priorityQueueID indicates which
    queue has reached the high watermark.
    @roseuid 38F3442F01B8 */
    oneway void signalHighWatermark (
        in octet priorityQueueID
    );

    /* This operation is a call event back to the PacketAPI client
    indicating that the queue has reach the low watermark.  If priority or
    multiple queues are being supported then this indicates that the sum total of
    all the queues has reached the low watermark.
    @roseuid 38F3446F025A */
    oneway void signalLowWaterMark (
        in octet priorityQueueID
    );

    /* This operation is a call event back to the PacketAPI client
    indicating that the queue has emptied.  If priority or multiple queues are
    being supported then this indicates that the sum total of all the queues has
    reached zero.
    @roseuid 38FE26CF02FA */
    oneway void signalEmpty ();
};

interface HfAleRealTimeAPI : MacPushPacket, connectionCommands {
};

#endif

```

10 UML.

This appendix includes the UML class diagrams for the Service Definition. The purpose for including these diagrams is to show the relationship between all the elements of the **HF ALE MAC API Service Definition**.

